11th International Conference on Customization and Personalization MCP 2024
The Power of Customization and Personalization in the Digital Age
September 25-26, 2024, Novi Sad, Serbia

GRAPH-BASED MULTI-AGENT ANALYSIS OF COMPONENT ASSEMBLY

Christian Becker [0009-0001-7562-6329], **Paul Christoph Gembarski** [0000-0002-2642-3445], **Roland Lachmayer** [0000-0002-3181-6323]
Leibniz University of Hanover, Institute of Product Development, Hanover, Germany

Abstract: *With rising customization demands, optimizing flexibility, productivity, and cost is essential. Traditional specialized processes are time-consuming and costly, especially when errors emerge during assembly. This paper introduces a multi-agent system (MAS) for automated assembly sequence generation. The system combines and complements existing sequencing approaches to allocate tasks to agents that analyze Computer-Aided Design (CAD) models, assess relations and dependencies, detect fasteners via image classification and utilize a graph-based approach to identify every potential assembly sequence. This facilitates early problem detection and workflow simplification for design engineers, ultimately enhancing efficiency and reducing costs in the product development cycle.*

Key Words: ***Assembly Planning (AP), Graph-based, Knowledge-based Engineering (KBE), Multi-Agent System (MAS)***

1. INTRODUCTION

Flexibility, productivity and cost optimization are becoming increasingly important as the demand for customization grows (Kumar, 2007). Specializing development, manufacturing and assembly processes for certain products requires time and expertise. To reduce the workload and pressure on developers and designers, it is necessary to optimize and support development and planning processes. Errors in assembly processes that are only discovered during assembly are particularly time-consuming, as the root of the problem may already lie in the requirements of the product and therefore the entire development cycle has to be run through again. In addition, the production and procurement of the individual components have already resulted in a high-cost factor. Digitalization in these areas allows early identification and implementation of improvements in models and processes (Chauhan *et al.*, 2023). In addition, companies can reduce development costs, shorten time-to-market and improve the quality of their products through automated and optimized processes.

This paper investigates the automated generation of assembly sequences from a 3D-model using a MAS to tackle the factors mentioned above. The agent system offers the possibility to automatically divide work packages into agents, to compare the processed information with other agents and to recognize conflicts, as well as to parallelize the processes (Jennings & Wooldridge, 1995). The CAD model is read out with different agents, the relations and dependencies of individual components are examined and then stored in a graph-based approach, which provides the possibility to work out possible (dis-)assembly sequences. The result is a scalable work simplification for the design engineers, with early identification of possible assembly sequences and potential problems. Therefore, multiple approaches are divided into separated agents and then fused in a MAS to figure out the possibilities of distributed systems in an assembly path-finding process.

To describe the path to the evaluated system, the paper is structured as follows: Section 2 presents the theoretical background and related work on multi-agent systems and automatic assembly planning. The methodological approach is then described in Section 3. The approach of a graph-based MAS for analyzing and subsequently finding an assembly path is explained in section 4 using an application example and discussed in section 5. A summary and description of further research can be found in Section 6.

2. THEORETICAL BACKGROUND AND RELATED WORK

The search for a solution to the flood of information has been an important area of research since the early 1990s. The possibility of collecting and processing data from various databases with minimal human intervention to solve a problem is still being sought and further developed today (Borghoff & Schlichter, 1998; Chauhan *et al.*, 2023).

The approach of splitting a problem into very small instances and then combining smaller programs to cover certain parts of the problem and additionally to learn from the processing cycle seems illusory at first. The first approaches to adaptive and intelligent agents, which are intended to support humans in almost all areas, laid the foundations (Eymann, 2003). In general, the agent can be described as a unit that generates an output from an input by means of information processing and performs defined tasks in the process (Ertel, 2016). The agents virtually represent people or machines and act independently

within defined limits. They are therefore classified as work facilitation and assistance for the user (Eymann, 2003).

In addition to the five main properties (observation, autonomy, mobility, communication and intelligence) described by Dostatni et al. (2013), according to Weyns (2010), agents have three services for interacting with the environment and accomplishing tasks. These include the function/action service, the perception service (sensors) and the communication service. With these services, the agent is able to analyze the environment, develop solutions and implement them. It also has an internal knowledge memory in which data collected during the process is stored.

Agents are generally classified as deliberative, reactive and hybrid agents. Deliberative agents are characterized by their explicitly representative database, the continuous cycle of observing, deciding and acting, as well as decision-making by inference. In contrast, reactive agents are situational and only perceive the current status of the environment and react directly to it (Eymann, 2003). The characteristics of these two classes are combined in the hybrid approach. Behaviors, planning and cooperation are divided into levels. This level model is known as the InteRRaP architecture. Furthermore, the agents have an internal knowledge base and an interface to the environment, the so-called 'world interface' (Bussmann *et al.*, 2004).

The BDI architecture by Geogeff and Rao (1992) is the best-known approach for deliberative agents. The architecture intends to implement mental properties and thus a conscience for the agent. The properties fundamentally include Beliefs, Desires and Intentions. These are extended by Uncertainty. From this, the agent can in turn derive and implement goals and plans (Borghoff & Schlichter, 1998; Eymann, 2003).

These architectures have been integrated into frameworks to simplify the programming of agent systems. SPADE (Smart Python Agent Development Environment) is a newer framework. The framework is Python-based and enables agents to communicate easily via XMPP, which also provides a simple interface to the user via instant messages. It also supports asynchronous, distributed and open systems. Since the framework was developed in Python, there is a broad basis for expansion and improvement, as Python is one of the most widely used programming languages in the field of AI (Artificial Intelligence), which is also supported by a large and active community (Palanca *et al.*, 2020). In addition, SPADE has an extension option with spade-bdi, an implementation of BDI agents that enables the reading and execution of ASL files programmed in AgentSpeak via an interpreter. In this file, plans can be defined in simplified language that trigger belief changes and intentions in the agent using the stored triggers (Palanca *et al.*, 2022).

In the literature, MAS are mainly used for path optimization and robot coordination in the assembly process to avoid collisions and find alternative assembly paths, due to the simple formalization of rules (Gembarski, 2020). There is no direct approach to CAD assemblies for assembly capability and the direct derivation of assembly sequences. Nevertheless, there are MAS approaches in design support and feature recognition. Chu et al. (2009) use a MAS to enable geographically separated work groups to work synchronously and to hide certain model details depending on the user's responsibilities. Another approach uses feature recognition agents for automatic adaptation to new situations (Fougères & Ostrosi, 2018). A similar adaptation is used by Plappert et al. (2023) in the course of the manufacturing restrictions of milling constructions resulting from feature recognition from graph-based methods for the automatic adaptation of individual CAD parts. They also present a possibility to communicate with the MAS through XMPP to check the need for chamfers and fillets in the CAD model.

For the most part, these approaches relate to individual parts, while assembly takes the models one level higher. Here, it is not only the relationships between individual surfaces and edges that are important, but also the relationships beyond the component. This implies contacting surfaces and functions of the individual surfaces and components. The 'assembly by disassembly' method is an important procedure for determining the assembly sequence. With its help, sequences and problems during assembly and disassembly can be easily detected by removing individual parts from an assembly piece by piece. The approach shows directly if the component is blocked or possible to disassemble by iterating different ways out of the assembly. In reverse, it is an indication of whether it can be assembled. Nevertheless, this method is very time-consuming due to the iteration of all possibilities. Ghandi and Masehian (2015) present five main categories of compiled methods for assembly planning. These include grid-based, graph-based, sampling-based methods, spatial decompositions and interactive approaches.

The grid-based and graph-based methods discretize a search space by reducing the action space with subspaces. The graph-based methods use a model of nodes together with weighted and directed connections to visualize relationships between components. This can be used to examine contact points or blocked paths in assemblies. For example, Belhadj et al. (2016) use this approach to find sub-assemblies with liaison graphs and Agrawal et al. (2014) use liaison and block graphs to detect collision-free assembly paths. Spatial partitioning is also used in other approaches to simplify the workspace. This is done by filtering processing or movement directions or by finding action spaces. Zhang et al. (2017) use the possible disassembly directions to set up interference matrices. The matrices can then be evaluated to check which component is blocked in which direction during disassembly. By comparing all directions and the number of interferences, a possible (dis-)assembly sequence can be found.

Based on the theoretical background and the related work of MAS and assembly path-finding, a MAS for automated assembly path-finding with graph-based methods is investigated as there is no approach in the literature. The system is designed to combine the advantages of distributed and graph-based assembly path planning solutions in a multi-agent system to save costs and time by alerting developers to further problems at an early stage and suggesting reliable solutions. The field of application is product development, with regard to a design review for the evaluation of assembly feasibility.

The following research questions were identified for the investigation:

- How are assemblies and the links between individual components structured in a CAD model, and how can information be extracted from a 3D assembly via an interface to the CAD system, processed and displayed in a graph-based manner?
- Which roles and agents are relevant in the process of finding assembly sequences, what problems or opportunities arise from their communication, and is a multi-agent system (MAS) suitable for the creation of assembly sequences?

3. METHODOLOGICAL PROCEDURE

Methods of the design process are used to support the analysis and construction of the system. In the following, the Multiagent Systems Engineering for Engineering Design (MaSE4D) method shown in Fig. 1 is presented, which is based on the MaSE method by Deloach et al. (2001) with elements of ROADMAP (Plappert, 2023). MaSE was developed specifically for heterogeneous MAS to create distributed, intelligent and robust applications. It covers the entire life cycle and provides an insight into the details of the system using the Unified Modelling Language (UML). This is intended to make communication and cooperation between agents simpler and more transparent (Deloach *et al.*, 2001).

The opposing triangles in the background represent the level of detail, from the broad system context to the inner knowledge model and then back to the outer overall system (Plappert, 2023).

The method is divided into the analysis and design phases. In the analysis phase, goals are first defined to generate use cases and consequently derive roles. This is achieved in two stages (Deloach *et al.*, 2001; Plappert, 2023):

- *Develop use cases*: Extracting and structuring system objectives from the requirements, visualized in a hierarchy diagram. Translating the objectives into use cases using various scenarios. Visualization and detailing in a use case diagram.
- *Refine roles*: Assign tasks to specific roles, organize them in a role model and depict the processing and dependencies in an interaction model.

The design phase translates the previously defined roles and goals into agents, a communication network and the final system. This phase consists of the following four levels (Deloach *et al.*, 2001; Plappert, 2023):

- *VI Operational*: Develop the required knowledge and identify communication protocols.
- *III Agents*: Derive agent classes from the roles that have functions to fulfill their roles. A class diagram is created from this.
- *II Management*: Graphically visualizes the communication between agents in the form of state machines or sequence diagrams. It should be noted that some tasks require communication with several agents and that a higher-level location is useful for better organization. Based on this, create an arbitrary architecture of the agents using UML components.
- *I Users*: Determine the final configuration (types and number of agents, platform, interfaces) and structure of the system, documented in a structure diagram. Subsequent implementation and testing by the users in the system context.

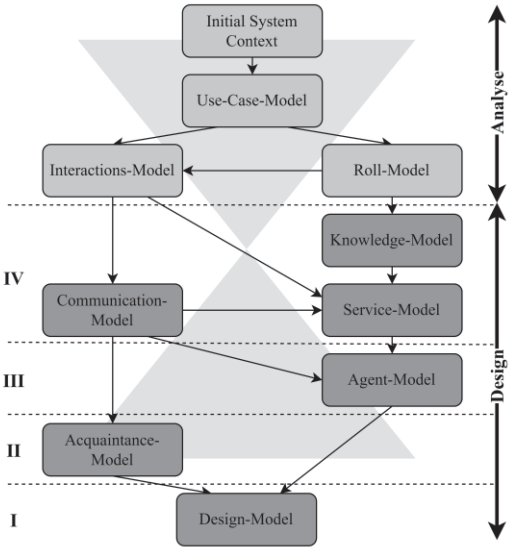


Fig. 1. *MaSE4D-Phases according to Plappert (Plappert, 2023)*

4. ASSEMBLY ANALYSIS WITH MULTI-AGENT SYSTEMS

4.1. Specification

The basic objectives of a support system are speed, ease of use and broad applicability, as well as robust solution finding and detailed documentation. To achieve these goals, target groups, application areas, company structures and possible inputs and outputs must first be analyzed.

In this case, a system is set up for the development or construction of an assembly. This implies that the realization of a concept in a 3D model does not mark the end of the development. The subsequent process steps are important instances for assessing the manufacturability and usability of the product. It therefore makes sense to integrate the planning, production and assembly cycles into the development process to recognise and eliminate problems at an early stage.

The solution is to provide engineers with applications that simplify their daily tasks and share some of their responsibilities. A key area is communication within the product life cycle, which often involves multiple expert opinions and can lead to misunderstandings, errors, or safety issues. Each change requires time-consuming validation or simulation loops, and some problems only become apparent during work or production planning. These challenges drive the development of tools that enhance safety, predictability, and integration into one

application. This application would simulate and compare databases to identify design issues, safety concerns, and guideline breaches, and facilitate communication with relevant departments. The system would log processes for transparency and traceability. When assembling modules, the design must consider restrictions from available machines, tools, and assembly structures, requiring the definition of use cases and roles.

Fig. 2 shows the use cases of the system. The user selects an assembly to be examined, which is read out by agents that also recognize connecting elements and patterns in the individual parts and use the relationships between the individual parts to show possible assembly sequences, visualize the results and then store them for documentation.

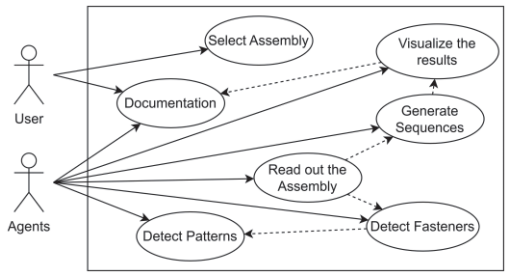


Fig. 2. *Use-Case diagram*

These use cases result in the roles and their dependencies shown in Fig. 3. On the one hand, the subdivision serves to subdivide areas so that areas of responsibility can be separated from each other and, on the other hand, to enable individual processes to be parallelized to save time and utilize computing power.

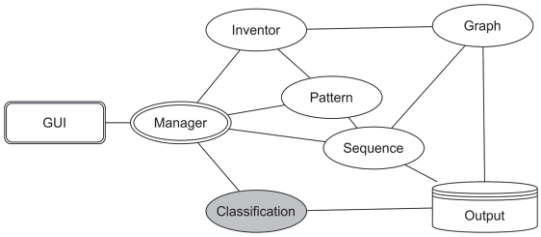


Fig. 3. *Examined Roles*

The master role is assumed by the manager, which is started via a Graphical User Interface (GUI), monitors the system and distributes the overall problem to subordinate instances. Such a structure enables better and simpler coordination of the agents and tasks. The Inventor role, which forms the interface to the CAD system and extracts the necessary information from the model, is subordinate. This role is separated to avoid overlapping access to the model. Furthermore, the pattern role attempts to find patterns in the individual parts to simplify the overall structure. The classification role is responsible for assigning certain classes to the individual parts to differentiate their use. This role is well suited to parallelization due to the consideration of individual components without reference to the assembly. For this reason, this task is separated into a role and reserved for the use of runtime-generated agents. The sequence role creates possible assembly sequences and the results are then visualized by the graph role. Outsourcing the creation of the graph makes it possible to prioritize the subsequent processes. The graph is largely used for sequence determination at the end of the process and can therefore operate in parallel with the other tasks. Furthermore, it is responsible for visualization in an image format, which is also only required for later archiving. Other agent results are stored directly in a results folder for archiving.

4.2. Architecture

The next step is to define the individual agents. The architecture of an agent is presented first. As shown in Fig. 4, the agent is programmed in Python using a MAS framework named SPADE (Palanca *et al.*, 2022). This already has predefined agent classes, which are extended with the BDI approach using the spade-bdi library.

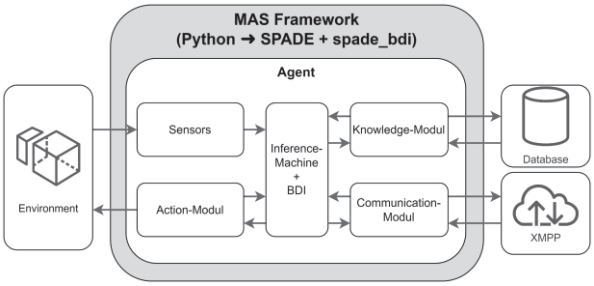


Fig. 4. *Architecture of an Agent according to* (Plappert *et al.*, 2022)

Each agent has five modules (Plappert, 2023):

- *Sensors*: Form the interface to the CAD-model and external influences by extracting information from the environment.
- *Inference-Machine*: The agent's brain, collects the incoming information and generates solutions according to its beliefs and intentions to achieve the desires and goals.
- *Knowledge-Modul*: The knowledge module manages external databases like thread-tables and stored graphs by extracting information from them or expanding them with the knowledge acquired during processing.
- *Communication-Modul*: The module uses an XMPP server to communicate and share acquired knowledge with other agents. Performatives defined by FIPA are used to categorize the intention of the message.
- *Action-Modul*: Once a solution has been found, it can be implemented in the environment or saved in a desired output using the action module.

In addition, each agent has an ASL file in which goals, intentions and desires are defined and the agent's beliefs are described and manipulated. The language AgentSpeak is used in the ASL file. This is a simple programming language that executes functions with trigger events. In the BDI view, the triggers are belief changes, goal realizations or test goals. However, belief changes in the ASL can also be triggered directly in the ASL.

4.3. Implementation

The basic task of an agent is to generate a possible output from an input. To achieve this, the agent must independently gather information and process it. This includes information or knowledge from databases,

opinions from users or other agents and, most importantly in the context of this work, information from the CAD assembly. This includes points, edges, surfaces and their relationships to each other. These are the properties that B-Rep and possibly CSG represent. (Stroud, 2011). Furthermore, the relationship of these structures and their dependencies between individual parts is also crucial.

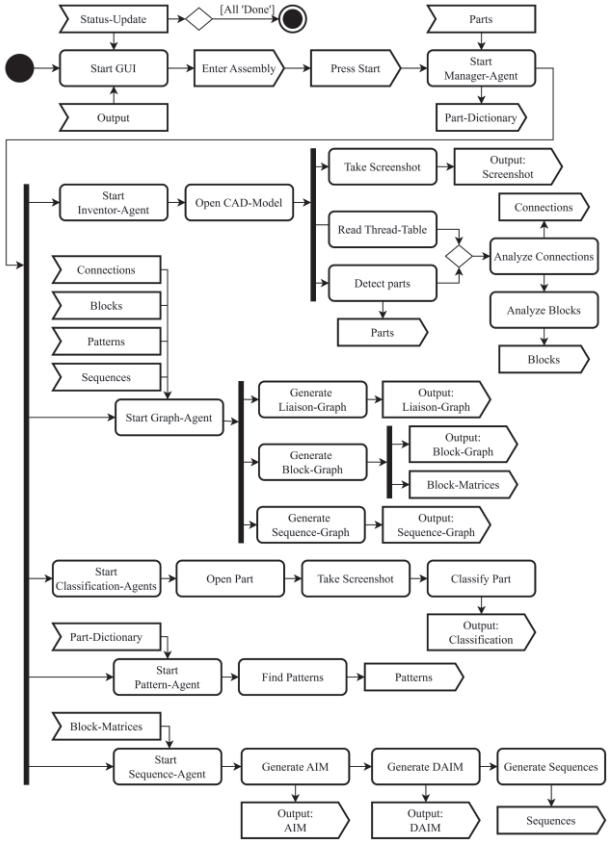


Fig. 5. *Activity diagram of the agents for finding assembly sequences*

Therefore, the most important instance in this system is the reading of the CAD model. The agents have to generate an image of the assembly themselves from the information read out and adapt it if necessary. The obstacle is the Python interface to Autodesk Inventor, as Inventor is designed for the 'Visual Basics for Application' (VBA) programming language and is therefore a Windows-based application. A VBA editor is therefore also implemented in Inventor (Ekins, 2007).

The Python library 'PyWin32' enables access to the 'Windows Application Programming Interface' (API) and thus to the API Object Model implemented in Inventor. It is important to address the correct class type when reading. By default, the 'Object' type is accessed, but this does not have all the attributes that the 'Edge' type has, for example. Here the Win32 library also contains a 'CastTo()' function, which assigns a new class type to a previously declared path. The entire structure and properties of the model can be read out using the interface created. Other object libraries can also be integrated via Python, such as 'Scripting' or 'mscorlib', which can retrieve objects from Windows libraries. This makes it possible to use Windows-based dictionaries and array lists. This is particularly important when accessing VBA functions via Python, as these only accept predefined object types that differ from the Python-based types.

In the following, the MAS is built up from the extracted roles, access to the 3D model and other functions for examining the model. The process and communication are also described here. To make it easier to follow the program flow is shown in Fig. 5 and the flow of information is visualised in Fig. 6.

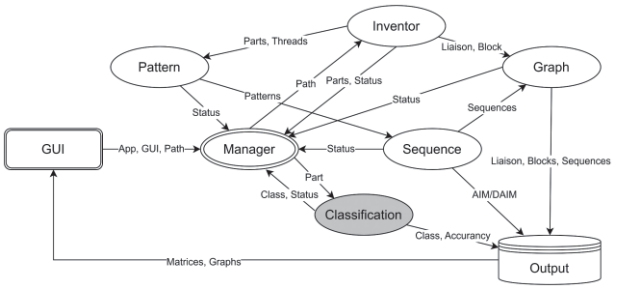


Fig. 6. *Information exchange between the different roles*

When the program is started by the user, a GUI appears in which the user can select the assembly to be examined. In addition, the interface has a status display to show the current progress of the program and also the status of the agents used. This status display is continuously updated by the manager agent as soon as an agent changes its status.

When an assembly is selected and the 'Get Started' button is pressed, the MAS starts. First, the manager agent is started by transferring the assembly path. This then starts the Graph, Inventor, Pattern and Sequence agents with the help of an integrated behavior for starting other agents.

Once the Inventor agent has started successfully, the Manager agent sends it the path of the assembly to be examined. With this path, the Inventor agent opens the respective assembly in Autodesk Inventor and starts to extract information from the model. First, a screenshot of the assembly is created for the archive. The agent then reads existing connections between the individual parts it contains. The collected individual parts are then forwarded to the Manager agent and the connections to the Graph agent.

The Inventor agent also reads a stored thread table. With this table, the agent can recognize threads when querying blocked paths and thus treat these affected areas accordingly. After processing, the function described above returns a dictionary with threads and the relationships of blocked individual parts. This dictionary is then forwarded to the Graph agent.

The Graph agent uses the connections to create a liaison graph, which shows the connection between the individual parts, and a block graph for each direction, which shows the parts blocked by the components. The Python package 'networkx' is used to translate the list of connections into a graph type. This graph is then plotted in a PNG file using the 'matplotlib' library. The block graphs are also converted into directed graphs and plotted using these packages. The block matrices generated when the graphs are created are then sent to the Sequence agent. At the same time, the Manager agent creates a

Classification agent for each individual part using the individual part dictionary.

The Classification agent creates a screenshot with the respective file path of the individual part. This image is analyzed with the previously trained image classifier with TensorFlow and assigned to a class, e.g. screws or washers. The classes and the accuracy are transferred to the Manager agent and the Classification agents can log off.

Once all individual parts have been classified, the Manager agent forwards the individual part dictionary to the Pattern agent. The Pattern agent searches for patterns of connecting elements in the individual parts using the function described above, which simplifies the further procedure. The results are then passed to the Manager agent, which implements them in the parts dictionary and forwards them to the Graph agent.

The Graph agent integrates the patterns into the already created graphs and merges several nodes and adopts their connections. The results are in turn exported to PNG files and the new matrices of the block graphs are transferred to the Sequence agent.

As soon as the Sequence agent has received the matrices, it creates the AIM (Assembly Interference Matrix) and DAIM (Directional Assembly Interference Matrix) from these matrices, both for the individual parts and for those with patterns. These are further processed with the sequence check, sequence paths are generated and the matrices are regenerated. The resulting AIMs are saved for the individual parts and matrices with patterns and the resulting dictionary is sent to the Graph agent.

The Graph agent creates a diagram from the sequence dictionary, which is exported in a PNG format. This makes it easy to trace the dis- and assembly sequences.

After all agents have changed their status to 'Done', the Manager agent collects the results and a new tab with the results appears in the GUI. The time required is also displayed in the status field.

4.4. Application Results

To better explain and show the functions of the MAS, a puzzle assembly as shown in Fig. 7 is used as an illustrative example. The assembly consists of a base plate, two interlocking puzzle pieces and five screws that connect these elements. In addition, the base plate has an edge that prevents the puzzle pieces from slipping without being screwed together.

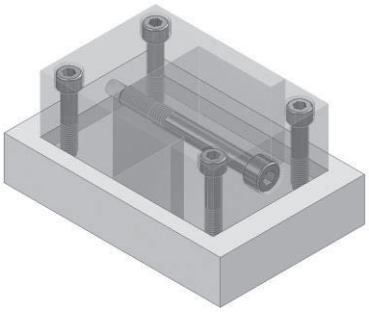


Fig. 7. *Example Puzzle-Assembly*

The assembly is loaded into the system via the GUI and then opened in Autodesk Inventor. The connections and blocked assembly paths are read out here. These are translated directly into the liaison graph and direction-dependent block graphs shown in Fig. 8. In this case, the liaison graph is very clear and depicts the existing connections between the individual parts with the help of a line between the respective nodes.

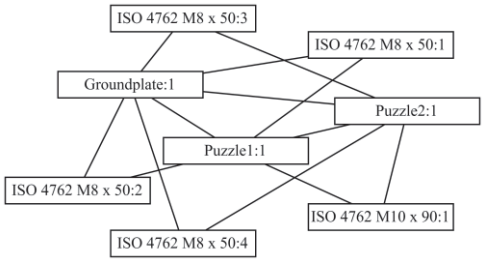


Fig. 8. *Liaison-Graph of the Puzzle-Assembly*

The block graphs are created for six mounting directions predefined in the program. It is important to note that the arrow represents '...is blocked by...'. An example of these block graphs is shown in Fig. 9. The graph shows the blocking components in the negative X direction. It can be seen, for example, that the component 'Puzzle1:1' blocks the screw 'ISO 4762 M8 x 50:2' in this direction, but the screw also blocks the component. The situation is different between the screws 'ISO 4762 M8 x 50:1' and 'ISO 4762 M8 x 50:2', here only the first is blocked by the second.

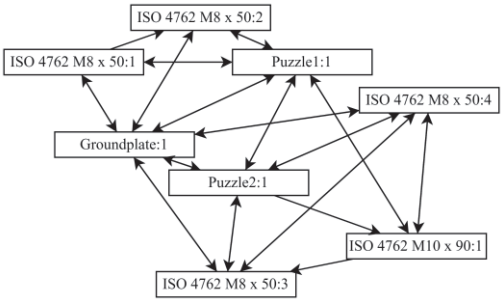


Fig. 9. *Block-Graph in -X of the Puzzle-Assembly*

To improve the overview of the graphs and save time, patterns of fasteners are searched for in the assemblies. To do this, all components are first classified by creating screenshots of the individual parts and assigning them to predefined classes using an image classifier. The results of the classification are shown in Table 1. It can be seen here that the screws are assigned to the correct class with a very high accuracy of almost 100%. The puzzle pieces are also assigned to the correct class. Only the Groundplate is not classified correctly, as the image classifier looks for similarities in the color pixel pattern of the training data and the images of the bearings fit better here. In this case, however, this error has no further effect, as only the connecting elements have any further impact.

Table 1 *Classification Results of the Puzzle-Assembly*

Name	Class	Accuracy in %
Groundplate:1	Bearing	98.599
Puzzle1:1	Other	99.422
Puzzle2:1	Other	80.716
ISO 4762 M8 x 50:1	Screw	99.999
ISO 4762 M8 x 50:2	Screw	99.999
ISO 4762 M8 x 50:3	Screw	99.999
ISO 4762 M8 x 50:4	Screw	99.999
ISO 4762 M10 x 90:1	Screw	99.995

The MAS can use this information and other information about position and directions to recognize patterns of connecting elements. This can be seen in Fig. 10, which shows the updated block graph in the negative X direction with patterns. It can be seen that four of the screws are grouped as they point in one direction and lie on one plane. This indicates that they are assembled and disassembled in the same way.

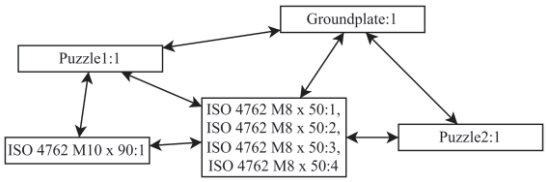


Fig. 10. *Block-Graph in -X of the Puzzle-Assembly with Patterns*

The AIM is then generated from the block graphs, which represents the dependencies of the blocking individual parts in a matrix. The zero stands for no blocking and the one for blocking in the specified direction. The AIM is then transformed into the DAIM shown in Table 2. This matrix shows how many obstructions the respective individual part has in the specific directions. The generated matrices are used to generate the possible sequences for disassembly. If there is a zero, as in the DAIM shown, the component can be removed and then the AIM and DAIM must be updated for each removed component. This is carried out for each possibility (for each zero) and then visualized.

Table 2 *DAIM of the Puzzle-Assembly with Patterns*

		-X	-Y	-Z	X	Y	Z
A	**Groundplate:1**	3	1	3	3	3	3
B	**ISO 4762 M10 x 90:1**	3	3	0	3	2	2
C	**ISO 4762 M8 x 50:1, ISO 4762 M8 x 50:2, ISO 4762 M8 x 50:3, ISO 4762 M8 x 50:4**	4	3	3	4	0	3
D	**Puzzle1:1**	4	2	3	4	2	3
E	**Puzzle2:1**	4	2	3	4	2	3

Fig. 11 shows a section of the assembly sequences described above. Starting from the assembly, the branches represent alternative disassembly paths and thus the zero in the DAIM. The boxes contain the components to be disassembled and the possible directions. At the same time, the assembly options can be read out by reversing the graph.

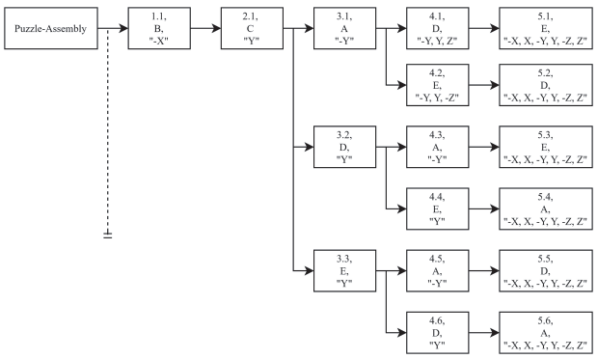


Fig. 11. *Assembly-Sequences of the Puzzle-Assembly*

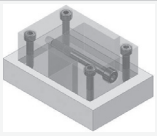
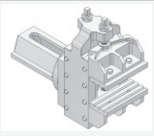
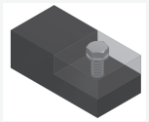
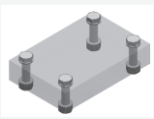
The steps that the MAS goes through to find a solution are continuously documented, output and archived in a text file. An excerpt of this log is shown in Fig. 12. This shows that each agent logs its steps from the start, through information received, its processing, the forwarding of the information and the status reports to the manager agent. The MAS required 107.1 seconds for the entire process of examining this assembly.

```
graph@agents.kl-hannover.de: Got Blockings from inventor@agents.kl-hannover.de.
graph@agents.kl-hannover.de: Generating Block-Graphs.
graph@agents.kl-hannover.de: Succesfully generated Block-Graphs.
graph@agents.kl-hannover.de: Sending Blockings to sequence@agents.kl-hannover.de.
manager@agents.kl-hannover.de: Got status-update from graph@agents.kl-hannover.de.
sequence@agents.kl-hannover.de: Got Dictionaries from graph@agents.kl-hannover.de.
sequence@agents.kl-hannover.de: Start generating AIM and DAIM.
sequence@agents.kl-hannover.de: Generated AIM and DAIM.
```

Fig. 12. *Excerpt of the Agent-Protocol*

In addition to the assembly described in detail above, other models are tested. To facilitate comparison, the differences in the number of components and patterns, as well as the time required, the classification success rate and the external dimensions are summarized in Table 3.

Table 3 *Distinguishing Features of Assemblies*

Index	**1**	**2**
Assembly		
Dimension [mm]	100x70x150	132x265x327
Parts	8	33
Patterns	1	3
Elements	5	24
Classification	7 of 8	23 of 33
Time [s]	107,1	3602,7
Index	**3**	**4**
Assembly		
Dimension [mm]	40x37x80	100x51x150
Parts	4	17
Patterns	1	2
Elements	3	3
Classification	4 of 4	17 of 17
Time [s]	50,4	305,7

The comparison shows that assemblies with more individual components and more complex geometries require more time. This is to be expected due to the additional analyses of the individual components and the increased computational requirements of the intersection of colliding components. It is also noticeable that the assemblies that are minimized by pattern recognition require much less time. The advantage here lies in the faster sequence processing. However, the size of the assembly is also relevant, as longer paths lead to a higher number of iterations during interference analysis.

The success rate for classifying simple parts is very good for the highly simplified model that was used. However, for more sophisticated components such as assembly 2, there is a clear need for improvements in this area. The model is designed exclusively for simple geometries and existing standard components.

5. DISCUSSION

The program enables the automated detection of connections between individual parts in an assembly. To do this, a model is loaded into Autodesk Inventor via a GUI and is read out. In addition, an image classifier is used to detect connecting elements, which are then analyzed and combined using pattern recognition. The information is used to generate a liaison graph to visualize the connections and direction-dependent block graphs for component-dependent blocked (dis-)assembly paths.

The AIM and DAIM, matrices that map the detected blockages, are derived from these graphs. Possible (dis-) assembly sequences are extracted from the DAIM, the AIM and DAIM are updated and then visualized. The system was set up with BDI agents that communicate with each other and jointly lead to a solution. The GUI is used for simple input and clear output of the results. The results are also stored in a folder together with the agents' log.

Simple and fast access is guaranteed for an overview and operation is made possible by a simple GUI. The system is easy to maintain and expand thanks to the division into individual agents. New functions can be easily implemented in the agents and the call can be realized via triggers in the ASL.

5.1. Interface Inventor

The PyWin32 library makes it easy to load and read the model in Autodesk Inventor. Unfortunately, the interface is quite slow with larger assemblies and components with complex shapes that require many connections of surfaces. Various internal Inventor functions are accessed in the work, which also operate via this interface. In addition, the interaction function offers a mapping of the interacting bodies, which often fails or takes more time with more complex geometries. It would be better to have a function that also checks the overlap, but only checks the overlap for required components, such as threads. Another point is the considered (dis-)assembly directions. These are currently limited to the six-axis directions, but this excludes many assemblies. The axes and surface normals of individual components would have to be checked beforehand to automatically recognize the required directions. In addition, the patterns should be included to save a double check and to analyze the complete pattern together.

5.2. Part Recognition

Classification must be brought forward for this, which will not be a problem in terms of the process. This leads to a further point for improvement, even if the recognition of the threads of fasteners works well, the recognition of the correct class of elements is not yet fully developed. Many elements are very likely to be assigned to the wrong class. For a start, it shows a possibility to detect fasteners, but a larger and more secure database is required for this. In addition, a highly simplified model was used with the TensorFlow model. This means that a database with far more images is required, which depicts elements from different directions well and contains different resolutions and representations to include user-defined representations. On the other hand, other models and training methods need to be considered to look at the features in more detail. It is also possible to create a feature database and include additional parameters and geometric properties of the component in addition to the machining feature approach of Zhang et al. (2022). Furthermore, the user can be consulted via a chat client, comparable to the mentioned chamfer and fillet conversation from Plappert et al. (2023), if the assignment is not clear, e.g. via the accuracy value. This is possible by using agents that communicate via an XMPP server. Chat clients or other messaging programs can easily be included here. This can also be used for further inquiries regarding patterns or similar purposes. On the one hand, it is used for recognition, on the other hand, it enables the standardization of connection elements in the assembly.

5.3. Sequencing

The sequencing of the assemblies runs smoothly with smaller groups like the example assembly, but the previously mentioned errors have a major impact. All possible collision-free assembly sequences can be determined quickly with a combination of ‘assembly by disassembly’-approaches using liaison and block graphs presented by Agrawal et al. (2014) and the DAIM of Zhang et al. (2017). This is an advantage because you can easily translate the relations from the graph into the AIM. At the moment, the paths still stop if there is no more zero in the DAIM. This means that the system does not solve this branch. For further consideration, the user can be consulted for a solution or the system itself must find a way to find a solution for components that cannot be dismantled. In this case, it makes sense to first look at the connecting elements to check in advance whether the connections can be made possible by additional mounting openings or holes. The system could generate various solutions for this and coordinate them with the user. It is also very important to ensure that all components of the assembly can be integrated without collision. This may not only be in relation to a single axis, but also to a combination of axes. One possibility could be the use of path algorithms to run through all eventualities. In addition, sub-assemblies dependent on the installation space and assembly can be detected in this way, which can be considered individually and create time in the assembly field through parallel processes. Furthermore, algorithms for path verification can also be used to optimize assembly processes. A database of available machines, tools and assignments can be used to analyze the time and costs required. Accordingly, it would be possible to automate the process chain and optimize the process at the same time. However, the influences of the environment, e.g. gravity, are also important in these analyses; this has not yet been considered and will have a very large impact on larger and unwieldy components. Components could fall or tip over when the connecting elements are loosened. The investigation of support points is crucial to prevent damage, accidents, and simplify the assembly process.

5.4. Visualization and Documentation

Another important point is the documentation of the decision-making process. All decisions must be comprehensible and accessible to the user. To this end, the connections, blockades and sequences are displayed in graphs. The results show that the visualization works, but

quickly becomes confusing with larger assemblies. Neo4J is an option for working with graphs. It offers a browser for visualizing and searching graphs simultaneously. In this browser, the nodes are dynamic and can be moved and hidden as needed.

5.5. Agents

Agent systems represent the main potential in this work. By using multiple instances, the clarity of the system is improved and can be steered towards specific optimization goals (costs, time, etc.) by manipulating the agent's internal decision-making. In addition, it is very easy to expand the system by dividing it into smaller clusters of functions. New functions can simply be integrated into the agent or new agents can be created. The challenge here is coordinating the agents and functions with each other. Furthermore, outsourcing into smaller programs offers the possibility of parallelizing processes and functions; thus both utilizing computing memory and saving a lot of time. They provide the option to create a digital replica of a process line, allowing for simulating and optimizing workflows. It also integrates work processes into development, identifying and potentially resolving design errors before production, if necessary.

6. CONCLUSION AND FURTHER RESEARCH

Based on the need for assistance systems for developers, this work used the MaSE4D methodology to develop a MAS that automatically generates assembly sequences from a 3D assembly.

The MaSE4D methodology was first presented to analyze system problems. Challenges in assembly planning were discussed, leading to defined objectives and requirements for the system. The problem was then divided into different roles, elaborating on the structure of an agent, assembly analysis, and result-finding functions. Finally, the integration of the agents into the system was described.

To test the MAS, an assembly and the results of the system were presented and discussed. Furthermore, various features of tested assemblies were compared with each other. The discussion showed that the system achieves good results, but that there is still room for improvement in some areas. The system is clear and easy to expand, especially due to the same structure and the event called by the ASL. In addition, it was possible to read out the CAD assembly, analyze the relations between the components and classify them, as well as arrange them in patterns and detect blocking components. The relations were successfully represented in graphs and transferred to the AIM and DAIM. With these matrices, it was possible to generate assembly sequences and also display them graphically.

Errors in the system were mainly due to misinterpretations during classification, which are attributable to the inadequate model. Furthermore, there is still room for improvement in the representation of the graphs due to the overlapping of nodes and edges in larger assemblies. And the most important point is the time required to find results. Especially for larger assemblies, the time increases exponentially due to the large number of connections and dependencies of the components. Nevertheless, the results show that this system has the potential to assist in the development process. Developers can identify and eliminate weak points in the assembly in the early stages of development without having to build a prototype and go through the assembly process. Even if there is still time to make improvements, the time and costs that the system can save are a real asset and relief in this area. At the same time, the system still offers many expansion options that can make the developers' work easier and make the processes safer, more cost-effective and situation-dependent.

The aim is therefore to further improve and expand this system. The analysis of the assembly via MAS must be further parallelized and additional information obtained from the CAD model. This information should be automatically enriched with further context-related information regarding production and assembly. This requires a well-structured data model. By integrating the machines, tools and process chains, the extent to which the system's development and production can be mapped by agents is checked. At the same time, the structure, coordination and communication of the agents is also an important development point. The agents must transparently demonstrate valid negotiation results, for which a solid communication network is required. Here, it is important to compare several approaches to how agents can discuss solutions among themselves to achieve an acceptable result for all parties.

7. REFERENCES

Agrawal, D., Kumara, S. & Finke, D.A. (2014) Automated Assembly Sequence Planning and Subassembly Detection. In: *64th annual conference and expo of the Institute of Industrial Engineers 2014: Montreal, Canada, 31 May - 3 June 2014.* Curran: Red Hook, NY, pp. 781–788.

Belhadj, I., Trigui, M. & Benamara, A. (2016) Subassembly generation algorithm from a CAD model. *The International Journal of Advanced Manufacturing Technology*, 87(9-12), 2829–2840. Available from: https://doi.org/10.1007/s00170-016-8637-x.

Borghoff, U.M. & Schlichter, J.H. (1998) Agentensysteme. In: Borghoff, U.M. & Schlichter, J.H. (Eds.) *Rechnergestützte Gruppenarbeit.* Springer Berlin Heidelberg: Berlin, Heidelberg, pp. 439–509.

Bussmann, S., Ishida, T., Jennings, N.R., Sycara, K. & Wooldridge, M. (2004) *Multiagent Systems for Manufacturing Control*. Springer Berlin Heidelberg: Berlin, Heidelberg.

Chauhan, S., Singh, R., Gehlot, A., Akram, S.V., Twala, B. & Priyadarshi, N. (2023) Digitalization of Supply Chain Management with Industry 4.0 Enabling Technologies: A Sustainable Perspective. *Processes*, 11(1), 96. Available from: https://doi.org/10.3390/pr11010096.

Chu, C.-H., Wu, P.-H. & Hsu, Y.-C. (2009) Multi-agent collaborative 3D design with geometric model at different levels of detail. *Robotics and Computer-Integrated Manufacturing*, 25(2), 334–347. Available from: https://doi.org/10.1016/j.rcim.2007.01.005.

Deloach, S.A., Wood, M.F. & Sparkman, C.H. (2001) Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(03), 231–258. Available from: https://doi.org/10.1142/S0218194001000542.

Dostatni, E., Diakun, J., Hamrol, A. & Mazur, W. (2013) Application of agent technology for recycling-oriented product assessment. *Industrial Management & Data Systems*, 113(6), 817–839. Available from: https://doi.org/10.1108/IMDS-02-2013-0062.

Ekins, B. (2007) *Unleashing Hidden Powers of Inventor with the API Part 1 of 4: Getting Started with Inventor VBA - "Hello Inventor!"*. Available from: https://download.autodesk.com/us/community/mfg/part_1.pdf [Accessed 22 December 2022].

Ertel, W. (2016) *Grundkurs Künstliche Intelligenz*. Springer Fachmedien Wiesbaden: Wiesbaden.

Eymann, T. (2003) *Digitale Geschäftsagenten*. Springer Berlin Heidelberg: Berlin, Heidelberg.

Fougères, A.-J. & Ostrosi, E. (2018) Intelligent agents for feature modelling in computer aided design. *Journal of Computational Design and Engineering*, 5(1), 19–40. Available from: https://doi.org/10.1016/j.jcde.2017.11.001.

Gembarski, P.C. (2020) Agent Collaboration in a Multi-Agent-System for Analysis and Optimization of Mechanical Engineering Parts. *Procedia Computer Science*, 176, 592–601. Available from: https://doi.org/10.1016/j.procs.2020.08.061.

Georgeff, M.P. & Rao, A.S. (1992) An Abstract Architecture for Rational Agents. In: *International Conference on Principles of Knowledge Representation and Reasoning*.

Ghandi, S. & Masehian, E. (2015) Review and taxonomies of assembly and disassembly path planning problems and approaches. *Computer-Aided Design*, 67-68, 58–86. Available from: https://doi.org/10.1016/j.cad.2015.05.001.

Jennings, N.R. & Wooldridge, M. (1995) Applying agent technology. *Applied Artificial Intelligence*, 9(4), 357–369. Available from: https://doi.org/10.1080/08839519508945480.

Kumar, A. (2007) From mass customization to mass personalization: a strategic transformation. *International Journal of Flexible Manufacturing Systems*, 19(4), 533–547. Available from: https://doi.org/10.1007/s10696-008-9048-6.

Palanca, J., Rincon, J., Carrascosa, C., Julian, V. & Terrasa, A. (2022) A Flexible Agent Architecture in SPADE. In: Dignum, F., Mathieu, P., Corchado, J.M. & La Prieta, F. de (Eds.) *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complex Systems Simulation. The PAAMS Collection.* Springer International Publishing: Cham, pp. 320–331.

Palanca, J., Terrasa, A., Julian, V. & Carrascosa, C. (2020) SPADE 3: Supporting the New Generation of Multi-Agent Systems. *IEEE Access*, 8, 182537–182549. Available from: https://doi.org/10.1109/ACCESS.2020.3027357.

Plappert, S. (2023) *Wissensbasierte Entwurfsbewertung der Produktgestalt mittels Multi-Agentensystemen*. Available from: https://doi.org/10.15488/14018.

Plappert, S., Becker, C., Gembarski, P.C. & Lachmayer, R. (2022) Feasibility Evaluation of Milling Designs Using Multi-Agent Systems. *Proceedings of the Design Society*, 2, 763–772. Available from: https://doi.org/10.1017/pds.2022.78.

Plappert, S., Becker, C., Gembarski, P.C. & Lachmayer, R. (2023) Scalable BDI-based Multi-Agent System for Digital Design Reviews. *Procedia Computer Science*, 225, 3593–3602. Available from: https://doi.org/10.1016/j.procs.2023.10.354.

Stroud, I. (2011) *Solid Modelling and CAD Systems: How to Survive a CAD System*. Springer-Verlag London Limited: London.

Weyns, D. (2010) *Architecture-Based Design of Multi-Agent Systems*. Springer Berlin Heidelberg: Berlin, Heidelberg.

Zhang, H., Zhang, S., Zhang, Y., Liang, J. & Wang, Z. (2022) Machining feature recognition based on a novel multi-task deep learning network. *Robotics and Computer-Integrated Manufacturing*, 77, 102369. Available from: https://doi.org/10.1016/j.rcim.2022.102369.

Zhang, W., Ma, M., Li, H. & Yu, J. (2017) Generating interference matrices for automatic assembly sequence planning. *The International Journal of Advanced Manufacturing Technology*, 90(1-4), 1187–1201. Available from: https://doi.org/10.1007/s00170-016-9410-x.

CORRESPONDENCE

Christian Becker, M.Sc.
Leibniz University of Hanover
Institute of Product Development
An der Universität 1
30823 Garbsen, Germany
becker@ipeg.uni-hannover.de

Paul Christoph Gembarski, Dr.-Ing.
Leibniz University of Hanover
Institute of Product Development
An der Universität 1
30823 Garbsen, Germany
gembarski@ipeg.uni-hannover.de

Roland Lachmayer, Prof. Dr.-Ing.
Leibniz University of Hanover
Institute of Product Development
An der Universität 1
30823 Garbsen, Germany
lachmayer@ipeg.uni-hannover.de